REMARKS:

Claims 1-57 were presented for examination and were pending in this application. In an Official Action dated August 12, 2009, claims 1-57 were rejected. Applicants thank Examiner for examination of the claims pending in this application and address Examiner's comments below.

Applicants herein amend claims 1, 17, 19, 46, 56 and 57. These changes are believed not to introduce new matter, and their entry is respectfully requested. The claims have been amended to expedite the prosecution of the application in a manner consistent with the Patent Office Business Goals, 65 Fed. Reg. 54603 (Sept. 8, 2000). In making these amendments, Applicants have not and do not narrow the scope of the protection to which Applicants consider the claimed invention to be entitled and do not concede that the subject matter of such claims was in fact disclosed or taught by the cited prior art. Rather, Applicants reserve the right to pursue such protection at a later point in time and merely seek to pursue protection for the subject matter presented in this submission.

Based on the above Amendment and the following Remarks, Applicants respectfully request that Examiner reconsider all outstanding objections and rejections, and withdraw them.

Objection to the Title

The title was objected to as allegedly as not being descriptive. The title is amended herein to correct the informalities indicated in the Office Action. In particular, the title is amended to indicate that the zero-time context switching of f the present invention is enabled by the context numbers associated with individual instructions. This feature distinguishes the

present invention from conventional thread scheduling schemes. Accordingly, reconsideration and withdrawal of the objection to the title is respectfully requested.

Objection to the Drawings

The Draftsperson has objected to Figures 1-6 and 7D-14 stating that the scale of these Figures is not large enough to and that the reference characters have insufficient height.

The attached Replacement Drawings for Figure 1-16 increase the scale of these Figures as well as enlarge the reference characters included in these Figures. No new matter is added by these amendments.

Approval of the proposed drawing changes is respectfully requested. It is also respectfully requested that the Examiner explicitly indicate his approval thereof in the next official communication.

Response to Rejections Under 35 USC §112, First Paragraph

Claims 2-18, 46-55 and 57 are rejected under 35 USC §112, first paragraph as allegedly failing to comply with the written description requirement.

Specifically, the Office Action alleges that the claimed element of "responsive to a context number associated with an instruction controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices," as recited by claim 17, is not described in the specification in such a way as to reasonably convey to one skilled in the art that the inventors were in possession of the claimed invention at the time the application was filed. *See* Office Action dated August 12, 2009, page 3. Claim 17 is amended herein to recite, in part:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate

> available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the second thread state for execution by said pipelined processor.

Amended claim 17 more particularly recites that the hardware thread scheduler identifies a program thread to be allocated processor time and that a context number associated with an instruction included in the identified thread is used to retrieve data for executing the identified thread from one or more registers included in a set of data storage devices. The thread selection hardware and execution schedule are described throughout the specification, for example at page 19, line 4 to page 25, line 14 and at page 27, line 10 to page 28, line 15. Similarly, the context numbers, and their use, are also described throughout the specification, for example at page 16, line 15 to page 18, line 15. Additionally, it is respectfully noted that the thread selection hardware determines the set of data storage devices from which the instruction is retrieved by directly coupling registers included in the appropriate set of data storage devices to execution pipeline, preventing a delay in execution of the instruction. This is contrary to the Office Action's assertion that "Clearly, if an instruction's context number is being analyzed, and that number is in the pipeline with the instruction, then that instruction has already been retrieved and the context number has no influence on where that instruction is fetched from." *See* Office Action dated August 12, 2009, pages 3-4. As claimed, thread selection hardware in the pipelined processor, not the execution pipeline, determines the data storage devicewhich provides the instruction to the execution pipeline. This is further

described throughout the specification, for example at page 26, line 21 to page 28, line 15 and FIG. 8. Thus, claim 17 now more clearly recites subject matter that is described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventors had possession of the claimed invention at the time the application was filed.

The Office Action also alleges that the claimed element of "said execution schedule specifying that the processor should directly switch to the first thread state from the second thread state every first number of cycles and that the processor should directly switch to the second thread state from the first thread state every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as recited by claim 46, is not described in the specification in such as way as to reasonably convey to one skilled in the art that the inventors were in possession of the claimed invention at the time the application was filed. *See* Office Action dated August 12, 2009, page 4. Accordingly, claim 46 is amended herein to recite, in part:

> switching the processor directly from the first thread state to the second thread state without incurring a time penalty by decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule and responsive to a context number associated with an instruction included in the first thread state or in the second thread state identifying the first set of data storage devices or the second set of data storage devices, said predetermined fixed execution schedule specifying that the first thread state should be allocated processing time every first number of cycles and that said second thread state should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles.

Accordingly, amended claim 46 now more clearly recites that the predetermined fixed execution schedule allocates processor time unequally between a first thread state and a second thread state and that when there is a switch from the first thread state to the second

thread state, the execution pipeline is decoupled from the first set of data storage devices and coupled to the second set of data storage devices. Support for the unequal allocation of processor time between a first thread state and a second thread state is found throughout the specification, for example at page 20, lines 4-14, page 21, line 12 to page 22, line 25, page 23, line 14 to page 23 line 6. Hence, claim 46 now recites material that is described in the specification in such a way as to reasonably convey to one skilled in the relevant art that the inventors had possession of the claimed invention at the time the application was filed.

Therefore, for at least these reasons, Applicants respectfully submit that the specification contains "a written description of the invention…in such full, clear, concise and exact terms as to enable any person skilled in the art to which it pertains…to make and use the same" with respect to amended claims 17 and 46. *See* 35 U.S.C. §112 (emphasis added). As claims 2-16, 18, 45-55 and 57 variously depend from claims 17 and 46, the subject matter of claims 2-16, 18, 45-55 and 57 is also described by the specification in such as way as to reasonably convey that to one skilled in the relevant art that the inventors had possession of the claimed invention at the time the application was filed. Hence, reconsideration and withdrawal of the rejection of claims 2-18, 46-55 and 57 is respectfully requested.

Response to Rejections Under 35 USC § 103(a)

Claims 1, 29-33, and 42-43 and 45-47 are rejected under 35 USC §103(a) as allegedly being unpatentable over U.S. Patent No. 6,076,157 to Borkenhagen et al. ("Borkenhagen") U.S. Patent No. 6,374,286 to Gee et al. ("Gee"). This rejection is respectfully traversed.

Claim 1 recites:

> a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second

> program threads by causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule by accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction included in a program thread identified for execution by the predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. (emphasis added)

The hardware thread scheduler recited in claim 1 is configurable so the processor switches from execution of one thread directly to execution of another thread according to a predetermined fixed schedule. This beneficially provides a predictable execution time for threads, allowing each thread to be executed for a predetermined number of instruction cycles specified by the predetermined fixed schedule. Additionally, by switching directly from the first thread to the second thread, the hardware thread scheduler reduces the overhead associated with switching between threads, allowing more time for execution of instructions from different threads. *See* spec., page 25, lines 8-13; page 27, line 4 to page 28, line 9. To directly switch from the first thread to the second thread, a context number is passed with an instruction through an execution pipeline. The context number of an instruction included in the selected thread is used to determine one or more registers, such as context registers from which register values are loaded or to which values are saved, to execute an instruction from the selected thread.

As the context number identifies the collection of storage devices, such as registers, describing the current processor state, switching between different contexts is a matter of using a different context number. By using a different context number, the execution pipeline accesses storage devices associated with the different context number, allowing direct switching from a first context, or thread, to a second context, or thread, without

performing system maintenance functions before executing the second context, or thread. *See* spec., page 17, line 19 to page 18, line 7. Associating a context number with an instruction allows thread switching at an instruction-level without requiring system maintenance or processor configuration between threads, unlike Borkenhagen and Gee.

In contrast, Borkenhagen discloses a system and method for data processing in a multithreaded processor where "[t]he thread switch logic has a time-out register which forces a thread switch when execution of the active thread in the multi-threaded processor exceeds a programmable period of time." Borkenhagen, Abstract. The time-out register in Borkenhagen is used "to force a thread switch to the dormant thread after some time if no useful processing is being accomplished to prevent the system from hanging." Borkenhagen, col. 14, lines 49-51. This forced thread switch prevents a thread from "spinning in a loop unable to do useful work" because it is unable to acquire ownership of, or access to, a necessary resource. Borkenhagen, col. 14, lines 31-34. Thus, the time-out register in Borkenhagen does not specify the processing time allocated to the first and second threads, but rather specifies a maximum time of inactivity before forcing a thread switch.

As admitted in the Office Action, Borkenhagen does not disclose "a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles." Additionally, Borkenhagen makes no

disclosure of "accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction included in a program thread identified for execution by the predetermined fixed schedule," as there is no disclosure or suggestion of context numbers associated with various instructions within threads.

Gee does not remedy the deficient disclosure of Borkenhagen. Rather, Gee discloses a single Java embedded microprocessor (JEM) to execute multiple Java Virtual Machines (JVMs) by assigning each JVM a fixed area of memory and a fixed amount of time in which to operate. After the fixed amount of time, the JEM invokes a master JVM which performs system duties and subsequently invokes a different JVM. Gee, col. 3, lines 50-65. When switching between JVMs, Gee does not directly transfer control of the JEM from one context to another, but switches from the currently operating JVM to a "master" JVM during a time period called an "interstice." During the interstice, the master JVM performs housekeeping duties and handles system interrupts. Once the interstice completes, the master JVM starts a proxy thread associated with the next JVM to be operated. The proxy thread checks the status of the associated JVM to determine whether or not the associated JVM is ready for operation. Gee, col. 3, line 61-col. 4, line 3. To switch between virtual machines, such as two JVMs, the JEM executes thread control blocks (TCBs) and executive control blocks (ECBs) constructed to look like JAVA objects to simplify manipulation by JAVA software. Gee, col. 19, lines 40-59.

To implement switching, Gee discloses an ECB using a "piano roll" scheduler including a list of entries which are periodically relayed in a fixed order responsive to receipt of an interrupt signal. Gee, col. 20, lines 44-53. This disclosed "piano roll" scheduler

executes of threads based on priority level rather than according to allocation of processor resources to individual threads. Gee, col. 32, lines 14-20. The "piano roll" scheduler shown by FIG. 19 of Gee is designed so that each rows "contains a bit for each priority level" indicating "whether the associated priority level will be active during the piano roll." Gee, col. 32, lines 16-18. Thus, rather than allocate resources among different threads, Gee merely discloses the frequency at which different priority levels are executed. The priority levels included in the "piano roll" scheduler are distinct from individual threads, as the association of a thread with a priority level is variable. Thus, the allocation of processor resources is dependent upon how the threads are associated with priority levels, which is determined using a mechanism distinct from the "piano roll" scheduler.

Further, there is no disclosure in Gee of "causing thread-switching between execution of the first program thread directly to execution of the second program thread at a fixed time according to a predetermined fixed schedule by accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction included in a program thread identified for execution by the predetermined fixed schedule," as claimed. Rather than access one or more registers in a set of data storage devices based on a control number from a thread instruction, Gee merely retrieves a separate software process based on a priority level associated with various software processes.

Hence, the claimed invention discloses a predetermined fixed schedule "specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" which is

used to identify a thread for execution and then executing the identified thread by "accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction included in a program thread identified for execution by the predetermined fixed schedule." In contrast, Gee merely describes activation of various priority levels with no disclosure of how instructions for a thread are retrieved, much less a disclosure of using an instruction-specific context number to retrieve data for executing an instruction from a thread. When a different priority level is to be executed, an interstice is executed and then software processes associated with the priority level are accessed and executed. Therefore, claim 1 is patentably distinct from the cited references, both alone and in combination.

Similarly, claim 46 recites:

> switching the processor directly from the first thread state to the second thread state without incurring a time penalty by decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule and responsive to a context number associated with an instruction included in the second thread state identifying the second set of data storage devices, said predetermined fixed execution schedule specifying that the first thread state should be allocated processing time every first number of cycles and that said second thread state should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles. (emphasis added)

As discussed above, Borkenhagen discloses using a time-out register to force a thread switch when the currently executing thread is inactive or fails to produce a useful result for a defined number of cycles to prevent a background thread from spinning in a loop. Thus, the time-out register in Borkenhagen specifies a maximum number of cycles in which a thread can be inactive or fail to produce useful output before forcing a thread switch. The time-out register in Borkenhagen allows a thread to continue executing indefinitely and does not switch

threads until the executing thread fails to produce useful output or becomes inactive for a defined number of cycles. There is also no disclosure in Borkenhagen of a "context number associated with an instruction in the second thread state" which is used to decouple the execution pipeline from the first set of data storage devices and couple the execution pipeline to the second set of data storage devices. Thus, Borkenhagen fails to disclose "switching the processor directly from the first thread state to the second thread state without incurring a time penalty by decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule and responsive to a context number associated with an instruction included in the second thread state identifying the second set of data storage devices, said predetermined fixed execution schedule specifying that the first thread state should be allocated processing time every first number of cycles and that said second thread state should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" as recited in claim 46.

Additionally, Borkenhagen makes no disclosure of directly switching from a first thread state to a second thread state without incurring a time penalty. The Office Action cites Figure 4A of Borkenhagen as allegedly disclosing this claimed element. However, Figure 4A is a block diagram describing the architecture of the thread switch logic described by Borkenhagen. Borkenhagen, col. 6, lines 42-44; col. 9, lines 25-34. In describing switching between threads, Borkenhagen provides:

> In a multithreaded processor, there are latency and performance penalties associated with switching threads. This latency includes the time required to complete execution of the current thread to a point where it can be interrupted and correctly restarted when it is next

> invoked, the time required to switch the thread-specific hardware facilities from the current thread's state to the new thread's state, and the time required to restart the new thread and begin its execution. In order to achieve optimal performance in a coarse grained multithreaded data processing system, the latency of an event which generates a thread switch must be greater than the performance cost associated with switching threads in a multithreaded mode, as opposed to the normal single-threaded mode.
>
> The latency of an event used to generate a thread switch is dependent upon both hardware and software. Borkenhagen, col. 12, lines 23-39.

Hence, Borkenhagen expressly discloses a latency associated with switching between threads rather than switching "directly from the first thread state to the second thread state without incurring a time penalty by decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule and responsive to a context number associated with an instruction included in the second thread state identifying the second set of data storage devices," as claimed. Like conventional thread switching systems, Borkenhagen discloses a time penalty, or latency, associated with switching between threads which is mitigated by the claimed use of context numbers associated with instructions within a thread.

As disclosed above, Gee fails to remedy the deficient disclosure of Borkenhagen, but rather discloses software processes executed by the JEM to switch between various JVMs. While Gee discloses "piano roll" schedulers to specify thread selection order, during context switches between JVMs "interstitial" activity occurs which consumes a number of cycles for performing system housekeeping or resource allocation in preparation for executing a different JVM, as depicted in FIG. 14 of Gee. Gee, col. 23, line 65 to col. 24, line 13; FIG. 14. Additionally, the "piano roll" schedulers disclosed by Gee are configured according to priority level, rather than based on threads, so that each row "contains a bit for each priority

level" indicating "whether the associated priority level will be active during the piano roll." Gee, col. 32, lines 16-18. Thus, rather than allocate resources among different threads, Gee merely discloses the frequency at which different priority levels are executed. The priority levels included in the "piano roll" scheduler are distinct from individual threads, as the association of a thread with a priority level is variable. Thus, the allocation of processor resources is dependent upon how the threads are associated with priority levels, which is determined using a mechanism distinct from the "piano roll" scheduler.

Further, there is no disclosure in Gee of "decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule and responsive to a context number associated with an instruction included in the second thread state identifying the second set of data storage devices, said predetermined fixed execution schedule specifying that the first thread state should be allocated processing time every first number of cycles and that said second thread state should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as claimed. Instructions within a thread are not disclosed by Gee, which switches between various JVMs by executing different software processes, much less a context number associated with an instruction within the first thread state. Such instruction-specific data is not disclosed by Gee, which merely uses priority levels, each associated with multiple threads, to modify thread processing. Nor is there a disclosure of coupling or decoupling an execution pipeline to different sets of data storage device in Gee, which merely discloses execution of JVMs by a processor.

Hence, amended claims 1 and 46 are patentably distinct from the cited reference for at least the reasons described above.

As claims 29-32 and 42-43 and 45 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 29-32 and 42-43 and 45.

As claim 47 is dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claim 47.

Applicants respectfully submit that for at least these reasons claims 1, 29-32, 42-43 and 45-47 are patentably distinguishable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

Claims 2-4, 13, 16-17, 19-24 and 56-57 are rejected under 35 USC §103(a) as allegedly being unpatentable over U.S. Patent No. 6,542,991 to Joy et al. ("Joy") in view of "Using Horizontal Prefetching to Circumvent the Jump Problem," by McCrackin et al. ("McCrackin"). This rejection is respectfully overcome in view of the amended claims.

Claim 17 recites:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the

second thread state for execution by said pipelined processor. (emphasis added)

Similarly, claim 19 recites:

> responsive to a predetermined fixed schedule allocating processing time to the first program thread and to the second program thread, switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty by coupling the execution pipeline from the first set of data storage devices to the second set of data storage devices via the hardware thread selector responsive to a context number associated with an instruction identifying the second set of data storage devices. (emphasis added)

Switching threads between consecutive instruction cycles beneficially allows switching between one program context and another without incurring any time penalty. Switching from one thread to another between the end of an execution cycle before the beginning of a next consecutive instruction cycle beneficially allows switching to occur without the loss of any execution cycles. To switch between the first and second threads without incurring any time penalty, the claimed invention couples one or more registers included in a second set of data storage devices which are identified from a context number associated with an instruction included in the second thread state to the execution pipeline. The context number allows the pipelined processor to directly retrieve data from, or store data to, the storage device associated with the instruction. *See* spec., page 18, lines 1-15. Hence, the thread selection hardware included in the pipelined processor determines the set of data storage devices providing data to the execution pipeline responsive to a context number associated with the thread to be executed, such as the second thread.

In contrast, Joy describes conventional event driven thread scheduling in a multithreaded processor which efficiently uses processor resources when a thread is stalled. See Joy, col. 6, lines 21-24. That is, the multithreading in Joy is for efficient use of the

processor, to reduce "wasted cycle times resulting from stalling and idling." See Joy, col. 2, lines 17-19. The thread switch logic of Joy includes "multithreaded-type functionality in response to an exception condition." Joy, col. 15, lines 8-11. Further, Context switches in Joy "typically are made in response to interrupts, including hardware and software interrupts, both internal and external, of a processor," rather than resposnive to a predetermined execution schedule, as claimed. Joy, col. 14, lines 62-63; col. 15, lines 4-7. While Joy enumerates various examples of thread switches, there is no disclosure in Joy of "identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule" or of "responsive to a predetermined fixed schedule allocating processing time to the first program thread and to the second program thread, switching the pipelined processor from executing the first program thread to executing the second program thread" as claimed. There is no predetermined execution schedule used by Joy, which instead initiates thread switches responsive to the occurrence of various events such as L1 data cache miss stall signals, instruction buffer empty signals, L2 cache miss signals, thread priority signals, thread timer signals, interrupt signals or other triggers. Joy, col. 3, lines 55-62.

The thread select logic described by Joy does not use a predetermined execution schedule to allocate processor resources, as claimed, but receives a plurality of input signals which evoke a context switch and a thread switch based on the occurrence of various conditions. Joy, col. 16, lines 1-13. While claimed invention enables deterministic performance of threads based on the predetermined execution schedule, Joy does not identify threads for execution or switch between threads until one or more inputs are received by the

thread switch logic. As these events do not occur in a fixed timeframe or at specified intervals, processing time is not allocated based on a predefined execution schedule by Joy, but is dynamically allocated responsive to system conditions. Hence, unlike the claimed invention, Joy discloses a conventional thread-switching system where thread switches are performed responsive to the occurrence of various events, rather than performed according to a predetermined execution schedule. Joy, col. 16, lines 1-15. There is no execution schedule disclosed, or contemplated, by Joy which instead generates a thread identifier signal on-the-fly based on the occurrence of different events. Hence, the thread switching performed by Joy is not performed according to a predetermined execution schedule, but is performed responsive to different events.

Additionally, while Joy's system includes "fast thread-switching," where timing of the thread switching process is described in detail, there is no mention of "consecutive cycles" or "zero-time" switching, as claimed. Joy, col. 3, lines 28-56. The fastest thread switching described in Joy requires an overhead cycle. Joy, col. 16, lines 1-5. As part of this overhead, Joy discloses that "thread switch logic generates the TID signal with a thread switch delay or overhead of one processor cycle." Thus, Joy discloses a very small delay in switching, but nonetheless a delay associated with generating the TID based on input signals received by the thread switch logic, rather than according to a predetermined execution schedule. Even the most ambitious portions of Joy teach the need of an overhead of at least one processor cycle caused by Joy's need to, responsive to the TID, save and restore the processor state associated with the currently executed thread and the thread to be executed. Joy, col. 15, lines 1-7. To perform a context switch, Joy requires that states from the first thread are saved and assigning new states to the second thread. Joy, col. 6, lines 42-48. To

save and restore a processor state, a store operation and a load operation are executed to transfer the current state to a memory and load data from a memory to the processor. While Joy provides that the thread switch logic provides "fast, nanoseconds range, context switching," the context switching in Joy executes instructions to save and restore processor state in response to various events. Joy, col. 15, lines 55-58.

In contrast, the claimed invention modifies whether the execution pipeline receives instructions from the first or second set of data storage devices. Rather than switch instructions responsive to a generated signal, the claimed invention directly fetches instructions from different storage devices responsive to the execution scheduler. For example, page 26, line 15 through page 27, line 2 of the specification and Figure 8 describe one implementation of the claimed invention. As described in the specification, multiple storage devices, shown in Figure 8 as "Flash A Fetch," "Flash B Fetch," "Flash C Fetch," "Flash D Fetch" and "Shadow SRAM," provide instructions to an execution pipeline based on the output from a hardware thread scheduler, illustrated in Figure 8 as "Post Fetch Select." Hence, the execution pipeline is capable of executing instructions from different threads during each instruction cycle by accessing a different storage device in each instruction cycle based on the context number associated with an instruction included in a thread to be executed. For example, a context number in an instruction identifies the storage device which provides instructions to the execution pipeline, allowing the execution pipeline to directly retrieve data from the identified data storage device. An example of this zero-time context switching is described in the specification at page 18, line 1 to page 19, line 5. Hence, the claimed invention executes instructions from different threads by modifying the storage device from which an execution pipeline retrieves instructions, allowing each stage in

an execution pipeline to execute instructions from different threads, while Joy requires that threads be loaded into the processor for execution, introducing overhead into thread switching caused by the time necessary for loading and storing processor state information. Hence, the claimed invention uses the context number of an instruction in the thread to be executed to determine from which data storage device to retrieve instructions.

Joy discloses using thread switch logic to generate a thread switch signal allowing the processor to change the currently executing thread. Joy, col. 16, lines 9-14. The thread switch logic generates a thread identifier to access a thread. However, the claimed invention associates a context number with instructions in the threads, allowing modification of the data storage device coupled to an execution pipeline depending on the context number associated with an instruction within the second thread. This context number allows the execution pipeline, or stages within the execution pipeline, to directly retrieve data from, or store data to, a particular storage device including data for executing an identified thread. This instruction-level association with a storage device is not disclosed in Joy, which at most allows identification of a thread in its entirety using a thread identifier generated in response to the occurrence of one or more events.

Hence, Joy does not switch threads between consecutive instruction cycles based on "according to a predetermined execution schedule" by switching "from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the second thread state for execution by said pipelined processor." Rather, Joy

performs an accelerated conventional context switching operation responsive to thread switch logic receiving signals associated with various events by saving and restoring processor states. Joy, col. 7, lines 50-52. By " switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty by coupling the execution pipeline from the first set of data storage devices to the second set of data storage devices via the hardware thread selector responsive to a context number associated with an instruction identifying the second set of data storage devices," responsive to a predetermined fixed schedule which allocates processing time to the first program thread and to the second program thread, the claimed invention allows deterministic execution of different instructions from different threads by specifying what set of data storage devices include the instruction associated with the context number. This also allows different stages in the execution pipeline to operate in separate contexts based on the context number of the instruction residing in the different stages.

McCrackin fails to remedy the deficient disclosure of Joy. Rather, McCrackin discloses horizontally prefetching instructions across different instruction. McCrackin, pg. 1287, Abstract. In McCrackin, streams compete for a bus interface and instruction unit for prefetching and an execution unit for execution by a horizontal demand prefetching (HDP) processor. As more streams are added to the HDP processor, prefetch depth increases. McCrackin, pg. 1288, § II, col. 1. A stream control unit selects different streams for execution by the HDP processor accounting for priority levels of each stream. McCrackin, pg. 1289, § III, cols. 1-2.

However, McCrackin also fails to disclose "identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule," as claimed. Rather, McCrackin discloses that "priority resolution mechanisms" are used to retch and execute requests from status element finite state machines. McCrackin, pg. 1289, § III, col. 1. Further, McCrackin discloses that "the highest priority SEL requesting service is given first preference" so that "as long as one stream requires service, it can be selected by the SCU, regardless of its position in the SEL ring." McCrackin, pg. 1289, § III, cols. 1-2. Hence, McCrackin does not disclose a "predetermined execution schedule" for allocating processing time, but dynamically modifies stream execution based on which stream currently requests resources. In contrast, the claimed predetermined execution schedule allows deterministic execution of threads by allocating a specified amount of processor time to various threads. McCrackin allocates resources based on the need for resources of a particular thread. The "demand" basis for resource allocation disclosed by McCrackin is contrary to the claimed "predetermined execution schedule" for resource allocation.

Also, McCrackin fails to disclose thread selection hardware in the pipelined processor which "switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the second thread state for execution by said pipelined processor," as claimed. McCrackin does not associate a context number

with an instruction within a stream to identify the storage device associated with the instruction, but selects between complete streams for execution. The stream control unit distributes processing time among various streams based on the demand level of streams and verifies execution of streams with full prefetch registers. McCrackin, pg. 1288, § II, col. 1.

McCrackin makes no disclosure of the instructions within a stream, much less a context number associated with an instruction within the stream which identifies a set of storage devices for coupling to the execution pipeline. Coupling of storage devices to an execution pipeline responsive to a context number associated with an instruction in a thread is not disclosed or suggested McCrackin as threre is no instruction-level context number identifying storage devices associated with an instruction. Therefore, McCrackin also fails to disclose "switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty by coupling the execution pipeline from the first set of data storage devices to the second set of data storage devices via the hardware thread selector responsive to a context number associated with an instruction identifying the second set of data storage devices," as claimed.

While the claimed invention associates a context number with an instruction included in a thread, the combination of McCrackin and Joy at most allows for switching between threads based on a thread selection signal generated by a thread scheduler or stream control unit responsive to an event. The claimed instruction-level context number and predetermined fixed schedule are not disclosed by Joy or McCrackin. While the claimed invention identifies a thread then identifies a context number from an instruction within the thread, Joy and McCrackin rely on an identifier associated with a thread as a whole, rather than a context

number associated with individual instructions within a thread. The generation of a thread selection signal to select an entire thread is distinct from associating a context number with an instruction within a thread to identify data storage devices for executing the instruction. While the claimed invention allows each stage in an execution pipeline to operate in a different context based on the context number, a thread-level selection signal requires all stages in an execution pipeline to operate in the same context.

Hence, claims 17 and 19 are patentably distinguishable from the cited references, both alone and combination, so reconsideration and withdrawal of their rejection is respectfully requested.

As claims 2-4, 13,16 and 57 are dependent from claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claims 2-4, 13, 16 and 57.

As claims 20-24 and 56 are dependent from claim 19, all arguments advanced above with respect to claim 19 are hereby incorporated so as to apply to claims 20-24 and 56.

Accordingly, for at least the reasons set forth above, claims 2-4, 13, 16-17, 19-24 and 56-57 are patentable over the cited references, both alone and in combination.

Claims 5-12, 18 and 25-28 are rejected under 35 USC § 103(a) as allegedly being unpatentable over Joy and McCrackin in view of U.S. Patent No. 6,085,215 to Ramakrishnan et al. ("Ramakrishnan"). This rejection is respectfully overcome in view of the amended claims.

As claims 5-12 and 18 are dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claims 5-12 and 18. As claims

25-28 depend from claim 19, all arguments advanced above with respect to claim 19 are hereby incorported so as to apply to claims 25-28.

Ramakrishnan is cited to make up for Joy's and McCrackin's lack of "thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" limitation. Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The switching in Ramakrishnan, like in Joy, is conventional context switching involving "an associated overhead in invoking the new thread." (Ramakrishnan, col. 12, lines 61-62, see also, col. 13, lines 6-7 "avoid time consuming context switching"). Ramakrishnan does not remedy the deficiencies of Joy and McCrackin discussed above. Hence, the combination of Joy, McCrackin and Ramakrishnan still fails to disclose:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the second thread state for execution by said pipelined processor.

and

> responsive to a predetermined fixed schedule allocating processing time to the first program thread and to the second program thread, switching the pipelined processor from executing the first program thread to executing the second program thread between the end of an

> execution cycle and before the beginning of a next consecutive execution cycle without incurring a time penalty by coupling the execution pipeline from the first set of data storage devices to the second set of data storage devices via the hardware thread selector responsive to a context number associated with an instruction identifying the second set of data storage devices.

as variously recited by claims 17 and 29. Accordingly, for at least the reasons set forth above, claims 5-12, 18 and 25-28 are patentable over the cited references, both alone and in combination. Thus, Applicants kindly request withdrawal of these rejections.

The Examiner rejects claim 14 as allegedly being unpatentable over Joy in view of McCrackin and Borkenhagen. This rejection is respectfully traversed.

As claim 14 is dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claim 14.

Borkenhagen is cited to make up for the deficiency of "storing said second thread state of said processor during execution of said first program thread" in Joy. However, the combination of Joy, McCrackin and Borkenhagen still fails to teach or suggest the "between consecutive instruction cycles" switching recited in claim 17. The system disclosed in Borkenhagen also incurs the conventional "latency and performance penalties associated with switching threads." Borkenhagen, col. 15, lines 37-38. Borkenhagen explicitly discloses:

> In the multithreaded processor in the preferred embodiment described herein, this latency includes the time required to complete execution of the current thread to a point where it can be interrupted and correctly restarted when it is next invoked, the time required to switch the thread-specific hardware facilities from the current thread's state to the new thread's state, and the time required to restart the new thread and begin its execution. (emphasis added)

Borkenhagen col. 15, lines 38-46. Thus, Borkenhagen also fails to disclose:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the second thread state for execution by said pipelined processor.

Hence, Borkenhagen fails to cure the deficiencies of Joy and McCrackin, so for at least the reasons set forth above, claim 14 is patentable over the cited references, both alone and in combination. Thus, reconsideration and withdrawal of this rejection is respectfully requested.

Claim 15 is rejected as allegedly being unpatentable over Joy in view of McCrackin in further view of U.S. Patent No. 6,314,511 to Levy et al. ("Levy"). This rejection is respectfully overcome in view of the amended claims.

As claim 15 is dependent on claim 17, all arguments advanced above with respect to claim 17 are hereby incorporated so as to apply to claim 15.

Levy is cited to make up for the lack of "said first set of data storage devices comprises registers shared by a plurality of threads" in Joy and McCrackin. However, Levy discloses a method "for freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no mention of associating a context number with an instruction to modify the storage device coupled to the execution

pipeline, but merely discloses a configuration for a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. Levy, therefore, does not remedy the deficient disclosure of Joy and McCrackin by also failing to disclose:

> a hardware thread scheduler for identifying which of said program threads said pipelined processor executes and configurable to allocate available processing time of the pipelined processor among at least the first and second program threads according to a predetermined execution schedule controlling whether the execution pipeline retrieves the instruction from the first set of data storage devices or the second set of data storage devices;
>
> wherein said thread selection hardware in the pipelined processor switches from said first thread state directly to said second thread state between consecutive instruction cycles without incurring a time penalty, by connecting one or more registers included in the second set of data storage devices identified from a context number associated with an instruction included in the second thread state to the execution pipeline, in response to the hardware thread scheduler identifying the second thread state for execution by said pipelined processor.

Accordingly, for at least the reasons set forth above, claim 15 is patentable over the cited references, both alone and in combination. Thus, reconsideration and withdrawal of its rejection is respectfully requested.

Claims 34-41 and 48-55 are rejected as as allegedly being unpatentable over Borkenhagen in view of Gee in further view of Ramakrishnan. This rejection is respectfully overcome.

As claims 34-41 are dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claims 34-41. As claims 48-55 are dependent on claim 46, all arguments advanced above with respect to claim 46 are hereby incorporated so as to apply to claims 48-55.

Ramakrishnan is cited to make up for the lack of "a thread identifier for identifying at least one hard-real-time (HRT) thread and at least one non-real-time thread" in Borkenhagen

and Gee. However, Ramakrishnan simply describes a software scheduler that uses a round robin approach to thread scheduling using conventional context switching. See Ramakrishnan, col. 9, lines 9-10. The scheduler in Ramakrishnan also fails to disclose "a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," "accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction included in a program thread identified for execution by the predetermined fixed schedule," or "switching the processor directly from the first thread state to the second thread state without incurring a time penalty by decoupling the execution pipeline from the first set of data storage devices and coupling the execution pipeline to the second set of storage devices via the hardware thread selector at a fixed time according to a predetermined fixed execution schedule and responsive to a context number associated with an instruction included in the second thread state identifying the second set of data storage devices, said predetermined fixed execution schedule specifying that the first thread state should be allocated processing time every first number of cycles and that said second thread state should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles," as variously recited by claims 1 and 46. Hence, Ramakrishnan does not remedy the

deficiencies of the combination of Borkenhagen and Gee, so claims 34-41 and 48-55 are patentable over the cited references, both alone and in combination. Thus, reconsideration and withdrawal of their rejection is respectfully requested.

Claim 44 is rejected as allegedly being unpatentable over Borkenhagen in view of Gee and in further view of Levy. This rejection is respectfully overcome in view of the amended claims.

As claim 44 is dependent on claim 1, all arguments advanced above with respect to claim 1 are hereby incorporated so as to apply to claim 44.

Levy is cited to make up for the combination of Gee and Borkenhagen lack of "said first set of data storage devices comprises registers shared by a plurality of threads." However, Levy discloses a method "for freeing a renaming register, the renaming register being allocated to an architectural register by a processor for the out-of-order execution of at least one of a plurality of instructions." Levy, col. 3, lines 27-30. Levy makes no disclosure of "a hardware thread scheduler for identifying which of said program threads said processor executes and configurable to allocate available processing time of the processor among at least the first and second program threads by causing thread-switching at a fixed time according to a predetermined fixed schedule, said schedule specifying that the first thread should be allocated processing time every first number of cycles and that the second thread should be allocated processing time every second number of cycles, wherein said first number of cycles is not equal to said second number of cycles" and "accessing one or more registers included in the first set or the second set of data storage devices based on a context number associated with an instruction included in a program thread identified for execution by the predetermined fixed schedule," as recited by claim 1. Rather, Levy merely discloses a

configuration for a "processor with dynamic out-of-order instruction processing capability." Levy, col. 7, lines 18-19. Accordingly, for at least the reasons set forth above, claim 44 is patentably distinct from the cited references, both alone and in combination. Thus, reconsideration and withdrawal of its rejection is respectfully requested.

Conclusion

In sum, Applicants respectfully submit that claims 1-57, as presented herein, are patentably distinguishable over the cited references. Therefore, Applicants request reconsideration of the basis for the rejections to these claims and request allowance of them.

In addition, Applicants respectfully invite Examiner to contact Applicants' representative at the number provided below if Examiner believes it will help expedite furtherance of this application.

Respectfully Submitted,
NICHOLAS J. KELSEY, ET AL.

Date: February 12, 2010

By: /Brian G. Brannon/
Brian G. Brannon, Registration No. 57,219
FENWICK & WEST LLP
801 California Street
Mountain View, CA 94041
Phone: (650) 335-7610
Fax: (650) 938-5200
E-Mail: bbrannon@fenwick.com

20880/05093/DOCS/2194613.1